second field name referred to as longname. The type indicates the datatype of the content of the text node and attribute nodes as well as fields containing a group as previously discussed.

Shortname and longname are two field names of different character length that may be selectively used to identify each field. Typically, longname is a field name identifying the content of the field in plain language. Shortname is also a field name, however, the format is abbreviated to a short sequence of letters and/or numbers unique to the field. When longname is used for the field names, larger amounts of information content are present in requests and output messages. Conversely, shortname provides lesser amounts of information content. Accordingly, shortname provides shorter messages that may be transferred over communication channels faster and/or with smaller carrier bandwidths. In addition, the field names in shortname are encoded identifiers thereby providing an extra level of security.

In one embodiment, Message class 44 includes a static variable that is a mode debug flag. The mode debug flag provides the ability to select between using longname or shortname as the mode of operation. Depending on the status of the mode debug flag, longname or shortname may be used as the field names for constructing and parsing messages. This selection is available for self-describing data structures that may create inefficiencies in production due to long field names. Selection of longname or shortname, in this embodiment, may be accomplished without changes to the source code within the business services layer 16. Changes in the mode of operation without changes to the source code also extends to the custom application code within the subclasses of BusinessService class 48 described later in detail.

BusinessService class 48 provides an abstract definition of a business services application that use the business services layer 16 as a framework. BusinessService class 48 is a generic superclass that may be used to provide standardize access to the back-end systems layer 18 for servlets and other applications. A plurality of subclasses may be included in BusinessService class 48 to provide data retrieval functionality for the business services application. The subclasses each represent different custom application code responsive to at least one request. Exemplary custom application code for a brokerage related business services application, for

example, may include services such as OrderAdd, OrderChange, OrderCancel, InquireOrders, InquireChecking, Transfer and other brokerage related requests for data. For an insurance related business services application, for example, custom application code corresponding to requests may include CheckHealthRisk, ApplyLife, ChangeLife and any other data requests related to insurance.

The custom application code provides an interface with the back-end systems layer 18. This interface may extract data from the back-end systems layer 18 based on requests and provide the data to the business services layer 16. The custom application code may extract data as a function of the request parameters translated into an XML structure within the input message. In addition, the custom application code may return the extracted data as a response. The data within the response may be translated to an XML structure within the business services layer 16 and provided as an output message as previously described.

In the presently preferred embodiments, the output message is translated by the business services layer 16 from the DOM document to HTML, WML or XML-text. In other embodiments, the output message may be translated to any other presentation format, such as, for example, Microsoft™ Word, plain ASCII Text, etc. Translation is preferably based on a format identified by the request or some other mechanism associated with the request. In one embodiment, the presentation format is determined by querying servlet header parameters included with the request. The servlet header parameters may indicate the type of presentation format compatible with the corresponding delivery technology within the end-users system layer 12.

If, for example, the servlet header parameters indicate a Microsoft™ browser is used as the delivery technology, XML-text may be returned (given the ability of Microsoft Internet Explorer™ to display XML text). Alternatively, if for example, the servlet header parameters indicate an HTML or WML compatible browser, the output message may be translated to HTML or WML.

Translation of the output message from the XML based structure to an output presentation may be performed using the XSL script 52. The translation involves the XSL processor API and at least one XSL stylesheet. In one embodiment, an XSL sylesheet is available for translation of each presentation format available. For

example, the XSL stylesheet for HTML output is utilized to translate the DOM document to HTML presentation format.

During operation, the basic flow of processing when a request is received by the business services layer 16 proceeds as follows: (1) Determination of a request name; (2) translation of request parameters to an input message with at least one field; (3) execution of subclasses of BusinessService class 48 to extract data from the back-end systems layer 18 based on the input message; and (4) creation and translation of an output message with at least one field to a desired format.

FIG. 3 is a flow diagram illustrating operation of the embodiment of the business services layer 16 illustrated in FIG. 2. The below description of operation identifies specific methods and instances within the business services layer 16. The methods of this embodiment are associated with each of the previously described classes as described below and detailed in the computer program listing appendix filed herewith. In other embodiments, other instances and methods may be used to provide the same functionality.

Processing within the business services layer 16 begins at block 100 with the receipt of a request from the front-end systems layer 14.

Determination Of A Request Name

Upon receiving the request, a doGet method of ApiService class 42 is executed at block 102. The doGet method manages the entire execution of request processing by first identifying a request name at block 104. The request name is identified based on the value of the request name parameter contained within the request. The request name parameter of one embodiment includes one of a plurality of possible predetermined strings. The predetermined strings are preferably predefined to represent different requests selectable by the end-user systems layer 12 from the front-end systems layer 14. In addition, the predetermined strings correspond to subclasses within the BusinessService class 48. In other embodiments, the request name parameters may include integers, characters or any other technique for uniquely identifying different requests.

Using the identified request name, an instance of a corresponding subclass of BusinessService class 48 is instantiated at block 106. The corresponding subclass is